Санкт-Петербургский Государственный Университет

Прикладная математика и информатика
Нелинейная динамика, информатика и управление

Варивода Иван Алексеевич

Создание клиент-серверного приложения для планирования путешествий

Бакалаврская работа

Научный руководитель:
член-корреспондент РАН, профессор, д. ф.-м. н. Леонов Г. А.

Рецензент:
генеральный директор ООО "НМТ-Новые Мобильные Технологии" Оносовский В. В.

Санкт-Петербург
2016

SAINT-PETERSBURG STATE UNIVERSITY

Applied Mathematics and Computer Science
Nonlinear Dynamics, Computer Science and Control

Varivoda Ivan Alekseevich

Development of client-server application for travel planning

Graduation Thesis

Scientific supervisor:
Member (corresponding) of RAS, Professor, Dr. of Science Leonov G. A.

Reviewer:
CEO "NMT-New Mobile Technologies" Onossovski V. V.

Saint-Petersburg
2016

Оглавление

1. Введение

Мы живем в век информации – время неограниченного доступа к информационным ресурсам, время, в которое объем данных растет экспоненциально. Каждый день тысячи туристических компаний, сервисов по заказу отелей, бронированию автомобилей публикуют десятки тысяч новых предложений. Каждое из них может быть потенциально интересно определенному пользователю.

Избыточное количество источников и неструктурированность этих предложений в совокупности создает ситуации, в которых человек может проводить большое количество времени, подбирая для себя подходящие предложения различных компаний и сервисов и планируя свое путешествие самостоятельно.

Имея такое разнообразие данных, появляется необходимость в сервисе, который мог бы пользоваться всей функциональностью этих служб и агрегировать все их возможности в одном приложении.

На данный момент существует большое количество сервисов, способных находить и заказывать места в отелях, бронировать автомобили, рестораны, но наряду с этим туристические компании могут предложить лишь готовые туры, и клиент становится зависимым от предлагаемых условий.

Разрабатываемое приложение должно предоставить возможность легко планировать и осуществлять заказы необходимых услуг запланированного путешествия, исходя из личных предпочтений и финансовых возможностей, не покидая 1 сайта.

2. Постановка задачи

Целью бакалаврской работы является создание клиент-серверного приложения для интеграции с сервисами, необходимыми при планировании путешествий, агрегации их в одном месте и предоставление пользователю необходимой информации в удобном виде.

Для достижения этой цели был сформулирован следующий набор задач:

- выбрать оптимальные сервисы для сбора информации;
- изучить необходимые технологии для разработки сервиса;
- разработать общую структуру проекта;
- спроектировать базу данных;
- написать логику работы приложения;
- создать пользовательский интерфейс.

В результате система должна удовлетворять следующим требованиям:

- кроссплатформенная независимость;
- актуальность предоставляемой информации;
- простота и удобство интерфейса;
- сохранение информации о созданном туре.

3. Обзор существующих подходов

3.1. Основные каналы дистрибуции

С развитием Интернет к традиционным ”игрокам” в индустрии туризма добавляются информационные посредники, которые делают возможным прямой доступ потребителя к поставщикам туристических услуг. На заре развития этих новых каналов дистрибуции даже стало популярным мнение об исчезновении туристических агентств как ненужных посредников между потребителями и поставщиками туристических услуг.

На данный момент существуют следующие основные каналы дистрибюции туристического контента:

- GDS (Global Distribution System) - глобальные распределительные системы (Amadeus, Galileo, Sabre); [9]
- сайты-поисковики (http://wwww.skyscanner.net);
- сайты-авиакомпаний (www.czechairlines.com, www.wizzair.com);
- сайты-агрегаторы (www.tickets.ua).

3.1.1. Глобальные распределительные системы

Первые информационные системы для туристического бизнеса появились в середине прошлого века и представляли из себя сервисы, ориентированные исключительно на одну отрасль, задействованную в туристическом бизнесе - авиаперевозки. Изначально эти сервисы имели бедный функционал и были неудобны в использовании. Именно в то время сформировалась потребность в единых информационных центрах, где поставщики услуг могли бы регистрировать свои сервисы, а пользователи осуществлять поиск и бронирование авиабилетов. Такие информационные центры называются Global Distribution System [7] и пользуются спросом до сих пор.

Естественно эти сервисы эволюционировали. На сегодняшний день GDS поддерживают также бронирование отелей, аренду автомобилей, трансферы и многие другие услуги. Основные игроки на сегодняшний день являются такие компании как Amadeus, Sabre.

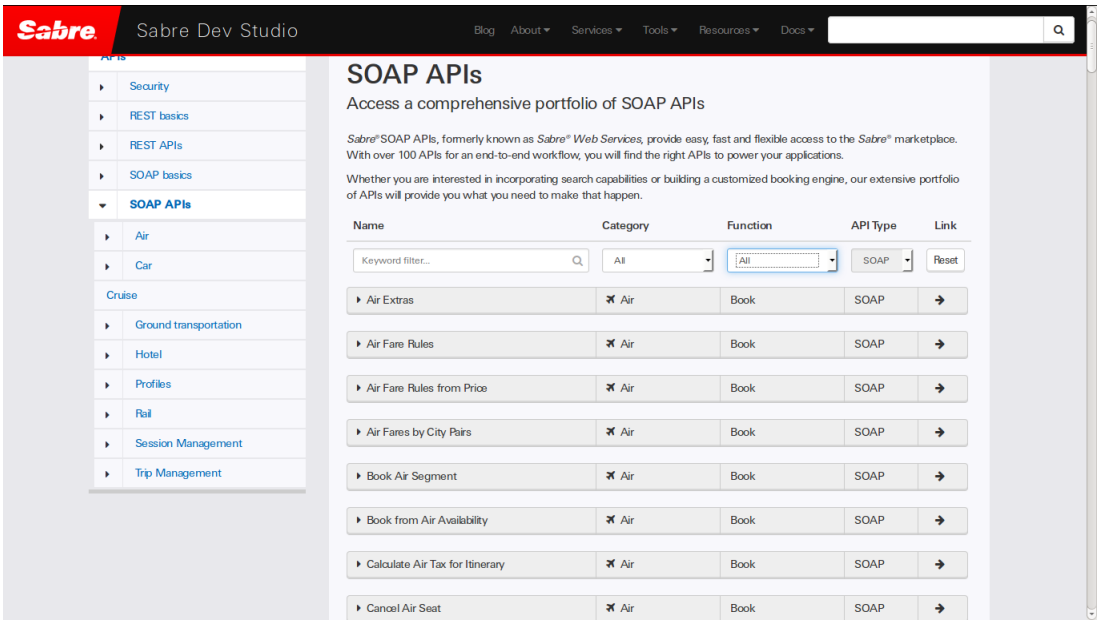

Рис. 1: GDS Sabre API

3.1.2. Сайты-поисковики

Поисковый сайт не предоставляет возможности покупки или бронирования билета. Он предназначен для отображения сравнительной

характеристики тарифов для различных авиакомпаний, отелей. Работа таких сайтов строится на использовании поисковых алгоритмов, позволяющих собирать данные от GDS, авиакомпаний, агрегаторов воедино.

При необходимости найти нужный рейс по разумной цене или забронировать гостиницу на несколько дней данный сервис подходит идеально, но при планировании различных деталей своего отдыха самостоятельно пользователю приходится пользоваться услугами различных сервисов по отдельности, что составляет определенные трудности.

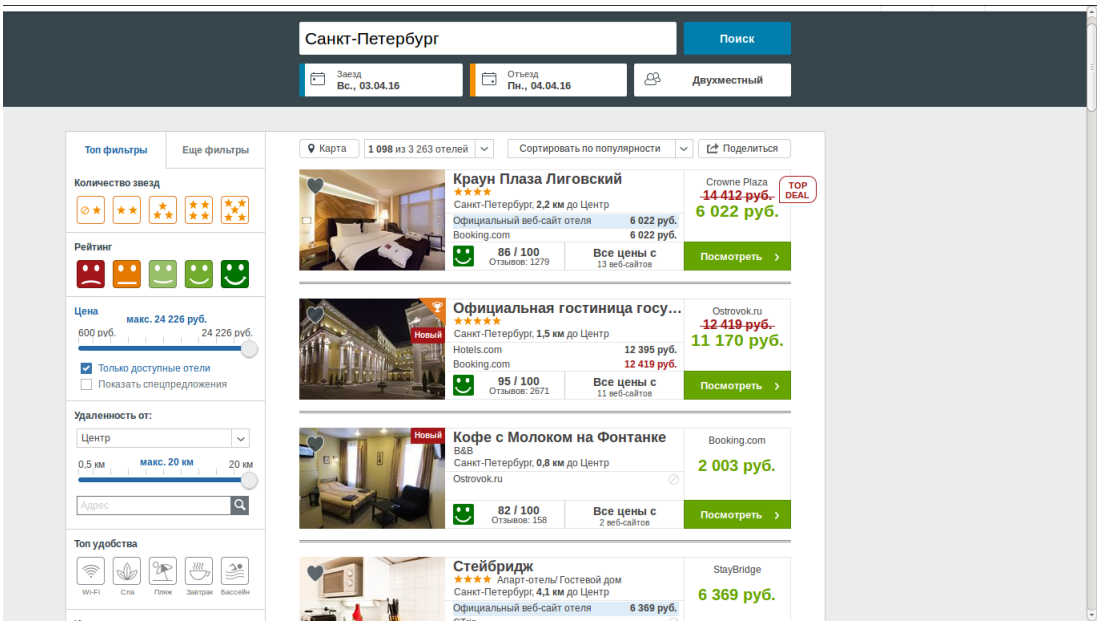

Рис. 2: Сайт-поисковик www.trivago.ru

3.1.3. Сайты-агрегаторы

Сайты-агрегаторы - веб-приложения, предназначенные для объединения данных из нескольких источников в один с единым пользовательским интерфейсом. Удобство этих сервисов заключается в том, что они позволяют не только осуществлять поиск необходимых услуг, но и допускают их приобретение в данной системе, что является неоспоримым преимуществом перед сайтами-поисковиками.

Однако большинство современных сайтов-агрегаторов позволяют лишь получать данные о готовых предложениях туристических фирм или же о доступных авиаперевозках, что опять же является недостаточным для

планирования независимого путешествия.

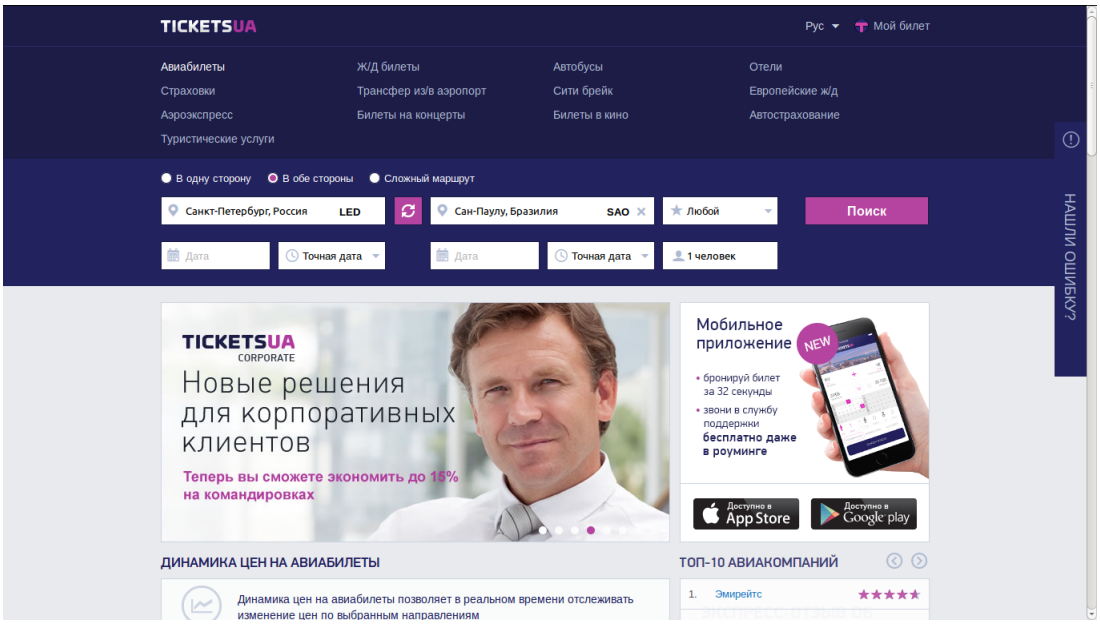

Рис. 3: Сайт-агрегатор www.tickets.ua

3.2. Выводы

После анализа этих решений были сделаны следующие выводы:

- Многие существующие решения не дают возможность приобретать услуги не покидая сайта;
- не все сервисы обладают широким выбором туристического контента;
- лишь небольшое число веб-приложений позволяют регистрироваться и сохранять основную информацию о своем путешествии.

Сервис, реализуемый в рамках данной работы, является веб-прило жением, представляющим собой систему для подбора персональных путешествий. Для использования приложения необходимо лишь наличие на устройстве современного браузера.

4. Предлагаемое решение

4.1. Выбор внешних сервисов

Для разрабатываемого приложения крайне важно разумно выбрать глобальную распределительную систему. На данный момент существует масса GDS со своими преимуществами и недостатками. Одними из самых известных являются:

- Travelport (Galileo, Apollo);
- Amadeus;
- Sabre;
- Pegasus Solutions.

4.1.1. Критерии существующих решений

Для рассмотрения и сравнения различных внешних сервисов необходимо определить ряд критериев, по которым будет выбираться основная GDS.

Будем использовать следующие критерии [7]

- Актуальность информации предоставляемой системой. Несвоевременная информация о статусе бронирования номера, например, может привести к потере клиента.
- Уровень детализации предоставляемой информации о туристических услугах. Уровень детализации позволяет сформировать общее впечатления клиента о данной услуге. Например, наличие фотографий номера или виртуальных туров по отелю помогает пользователю определиться с выбором.
- Расширяемость бизнес-правил. Система должна позволять проводить подстройку бизнес-правил и внешнего вида под требования компаний.

- Затраты на сопровождение и администрирование системы. Сложные в сопровождении системы значительно увеличивают затраты на внедрении и поддержание ее работы в штатном режиме.

- Возможности по интеграции со сторонними системами. Данные возможности позволяют значительно расширить круг поддерживаемых услуг.

- Охват рынка. Большой охват рынка позволяет сформировать компании широкий ассортимент услуг.

4.1.2. Анализ GDS

Для определения необходимой GDS проведем анализ следующих систем.

Sabre [6]

- Актуальность информации. Sabre использует интеграцию с собственной GDS Sabre для актуальности информации.

- Уровень детализации информации. Sabre позволяет просматривать изображения отелей.

- Расширяемость бизнес-правил, информационных потоков внутри системы.

- Возможность интеграции. Sabre предоставляет довольно развитый внешний API, позволяющий получать актуальные данные.

- Тестовый интерфейс. Данная GDS имеет богатый API для тестирования собственного приложения, что позволяет адекватно настроить систему и протестировать ее без использования реального сервиса.

- Бесплатный доступ к функционалу для тестирования.

Amadeus

- Актуальность информации.
- Уровень детализации информации. Например, e-Cruise позволяет потребителям просматривать виртуальные туры по кабинам судна[1]
- Расширяемость бизнес-правил. Система Amedeus Selling Platform содержит ряд возможностей по настройке внешнего вида системы под требования турагентства.
- Поддерживаемые группы пользователей. Продукты Amedeus рассчитаны почти на всех участников туристических услуг.

После анализа можно увидеть, что по основным критериям данные системы удовлетворяют потребностям приложения. Однако, Sabre также предоставляет API для тестирования разрабатываемого агрегатора, что послужило основной причиной выбора данного сервиса.

4.2. Выбор инструментов

4.2.1. Серверная часть

Для реализации серверной части были выбраны технологии, описанные ниже.

В качестве основного языка программирования было принято решение использовать Java [2]. Основополагающей причиной данного выбора послужило существование для этого языка различных библиотек и инструментов, позволяющих осуществлять быструю и качественную разработку веб-приложения.

При разработке серверной части, использовалась технология Java Enterprise Edition [2] . Этот набор спецификаций и соответствующей

[1]Amadeus launches e-Cruise in the midst of rising demand, URL: http://www.amadeus.com/uk/x165739.htm (дата обращения 21.03.2016

[2]Язык программирования Java, URL: http://docs.oracle.com/javase/8/docs/api/ (дата обращения 21.03.2016)

документации для языка java, описывающий архитектуру серверной части, обеспечивает масштабируемость приложения, целостность данных во время работы системы и гибкость при создании приложения.

Одним из преимуществ JEE является доступность данной технологии. На текущий момент существует большой выбор серверов приложений, поддерживающих эту спецификацию. Основными являются: GlassFish[3] компании Oracle, Apache TomEE[4].

В качестве сервера приложений для проекта было принято решение использовать Apache TomEE.

Для объектно-реляционного отображения java-классов используется фреймворк Hibernate [1]. Основными достоинствами данного фреймворка являются:

- Поддержка объектно-ориентированного подхода при отображении. Hibernate позволяет отображать Java-классы в соответсвии с ООП, что дает возможность сделать более гибким архитектуру приложения.

- Собственный язык для создания запросов Hibernate Query Language (HQL), упрощающий построение SQL-запросов и облегчающий процесс разработки.

- Высокая производительность. Фреймворк позволяет выполнять "ленивую" выборку информации из базы данных (lazy loading), что существенно экономит ресурсы и время.

За сборку проекта отвечает система сборки Maven [4]. Основными преимуществами данной библиотеки являются:

- Независимость от операционной системы. Файл проекта один и тот же для любой ОС.

- Управление зависимостями. Большинство современных приложений зависят от массы сторонних библиотек, которые в свою оче-

[3]GlassFish Server, URL: https://glassfish.java.net/ (дата обращения 21.03.2016)

[4]Apache TomEE, URL: http://tomee.apache.org/apache-tomee.html (дата обращения 21.03.2016)

редь, используют библиотеки различных версий. Maven дает возможность управлять такими сложными зависимостями.

- Интеграция со средами разработки. Система используется в почти всех современных IDE, в частности в Intellij Idea, что дает возможность настраивать параметры сборки и собирать проект не покидая IDE, что экономит время.

Для управления базой данных используется СУБД MySQL[5].

4.2.2. Клиентская часть

Для разработки клиентской части были использованы технологии, описанные ниже.

Основная клиентская часть написана с использованием HTML и CSS.

Технология Java Servlet Page 2.1 (JSP[6]) позволяет создавать веб-страницы, которые имеют как статические, так и динамические компоненты. Страница JSP содержит текст двух видов: статические исходные данные, которые в настоящем случае будут оформляться в формате HTML и JSP-элементы, конструирующие динамическое содержимое.

Также для упрощения работы с JSP используется набор тегов Java Server Pages Standard Tag Library (JSTL). JSTL - это расширение спецификации JSP, которое добавляет библиотеку JSP тегов для выполнения основных операций, таких как: условная обработка, создание циклов и поддержание интернационализации, что значительно экономит время разработки и разделяет бизнес-логику и визуализационную часть.

[5]СУБД MySQL, URL: http://www.mysql.ru/docs/ (дата обращения 21.03.2016)

[6]JSP 2.1, URL: http://www.oracle.com/technetwork/java/javaee/jsp/index.html (дата обращения 21.03.2016)

4.3. Архитектура приложения

4.3.1. Структура базы данных

База данных хранит информацию о клиенте, его предпочтениях поездки, и основные компоненты туристического контента. Структура базы данных имеет следующий вид.

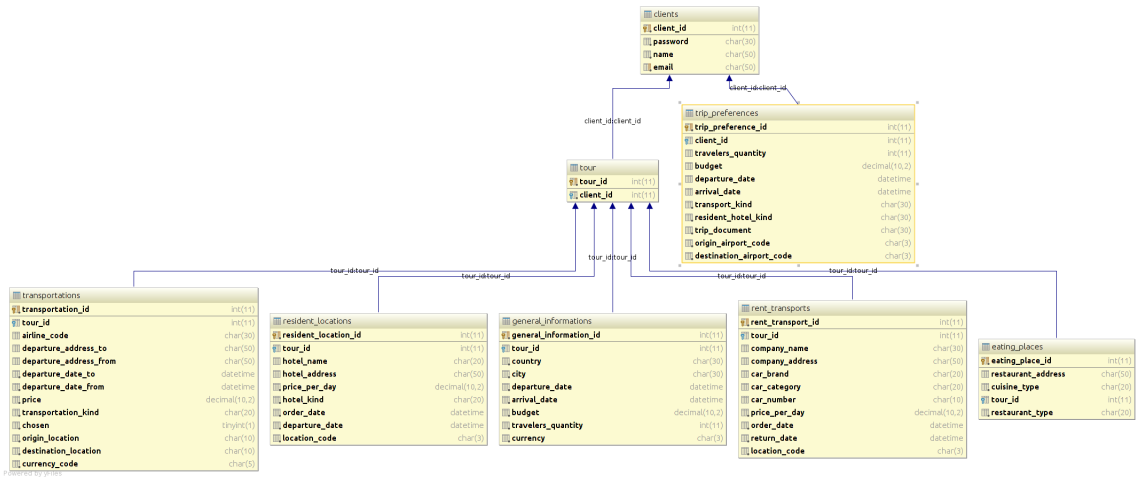

Рис. 4: Диаграмма базы данных

При проектировании данной системы использовались следующие паттерны.

4.3.2. MVC model

Model-View- (MVC) [8] - схема использования нескольких моделей проектирования. Данная схема позволяет разделить пользовательский интерфейс, взаимодействие с пользователем и модель приложения на три отдельные компонента, которые мало зависимы друг от друга.

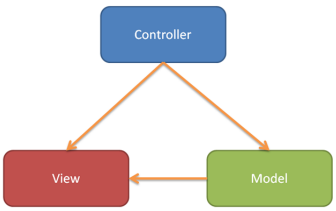

Рис. 5: MVC model

4.3.3. Модель

Модель представляет собой данные, с которыми оперирует приложение. Это могут быть как информация из базы данных, так и любая другая структура данных, описывающая некоторые объекты системы и их состояние.

В разработанной системе модель представлена следующими двумя пакетами:

- model.client
- model.tour

Пакет model.client содержит классы для хранения информации о клиентах и их предпочтениях. Следующая диаграмма демонстрирует классы с их свойствами.

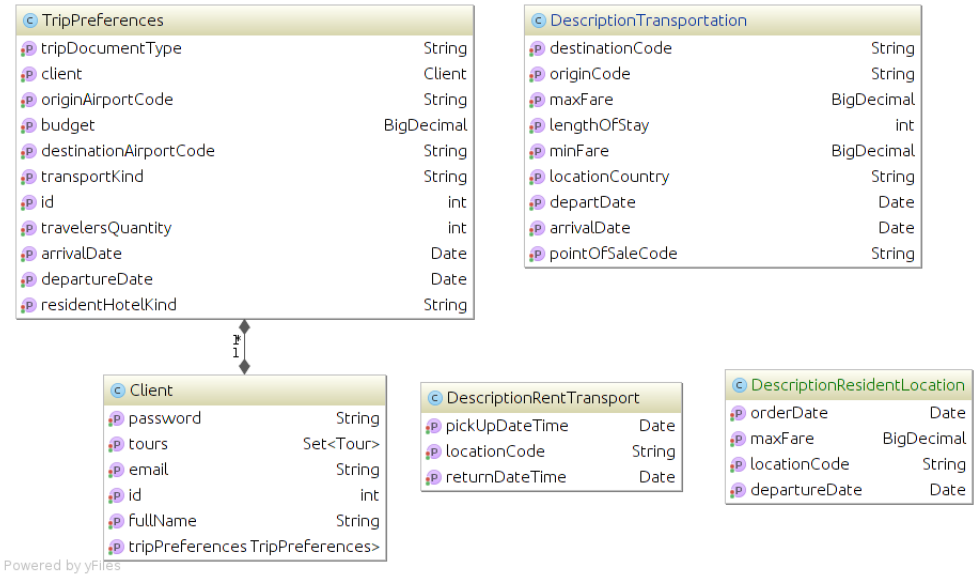

Рис. 6: Структура пакета model.client

Для хранения информации о туристическом контенте предназначен пакет model.tour. Ниже представлены классы, хранящие необходимую туристическую информацию.

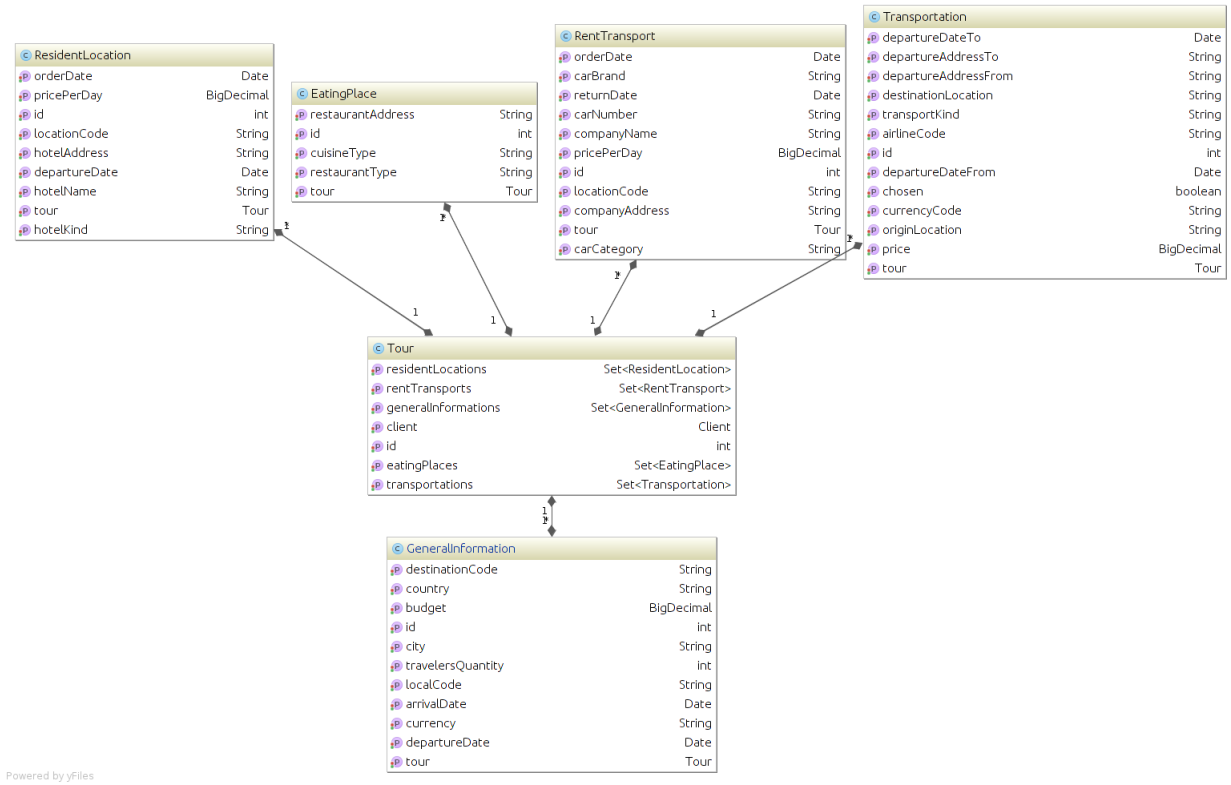

Рис. 7: Структура пакета model.tour

4.3.4. Вид

Вид представляет собой компонент системы для отображения состояния модели в понятном для человека виде. Важно отметить, что данный компонент не изменяет состояние объекта. В разработанной системе этот составляющая представлена набором jspx-документов и html-страниц.

4.3.5. Контроллер

Контроллер - средство, при помощи которого пользователи взаимодействуют с системой. Данный компонент также является управляющим элементом для обмена данными и сообщениями между видом и моделью.

Контроллер приложения можно разделить на несколько частей. Общая структура данного элемента отображена на диаграмме.

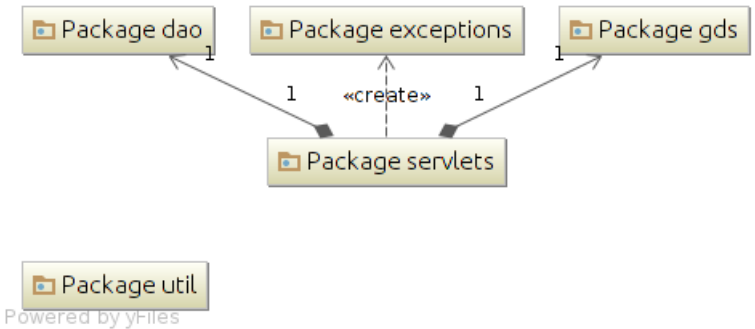

Рис. 8: Общая структура контроллера

- controller.dao - модуль, осуществляющий взаимодействие с базой данных;
- controller.gds - модуль, предназначенный для обмена информации между приложением и GDS;
- controller.servlets - набор специальных java-классов, отвечающий за взаимодейсвите системы с пользователем;
- controller.util - набор утилит, упрощающих работу основных сервисов;
- controller.exceptions - набор исключений для приложения.

4.3.6. Модуль взаимодействия с базой данных

При проектировании этого модуля было принято решение использовать DAO-паттерн (Data Access Object) [3]. DAO - это объект, который реализует необходимый для работы с источником данных механизм доступа. Преимущество данного шаблона заключается в том, что у него могут быть различные по природе источники данных, например репозиторий, бизнес-служба, обращение к которой осуществляется при помощи протокола CORBA, персистентное хранилище данных.
Использующие DAO бизнес-компоненты работают с более простым интерфейсом, который скрывает все детали реализации источника данных от клиента.

Поскольку при изменениях реализации источника данных представляемый DAO интерфейс не изменяется, этот паттерн дает возможность DAO принимать различные схемы хранилищ без влияния на клиентов или бизнес-компоненты. По существу, DAO выполняет функцию адаптера между компонентом и источником данных.
В разработанном приложении пакет controller.dao состоит из 3 частей.

- Набор абстрактных внешних интерфейсов, позволяющий выполнять операции сохранения, удаления, обновления любого объекта, представленного в пакетах model.client и model.tour

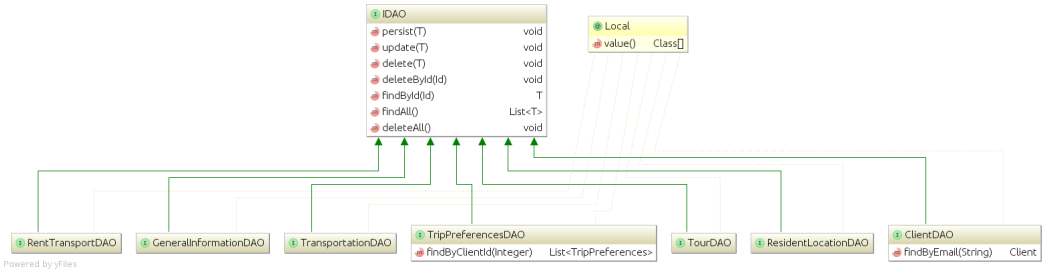

Рис. 9: Внешний интерфейс модуля

- controller.dao.util - набор утилит, позволяющий упростить взаимодейтсвие компонент данного модуля

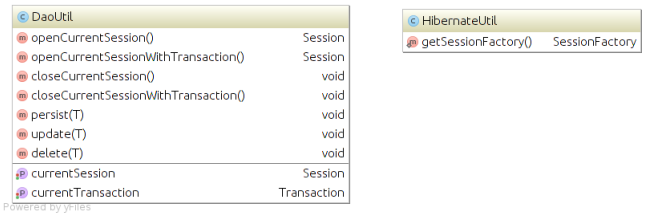

Рис. 10: Утилиты для работы с DAO

- controller.dao.impl - реализация внешнего интерфейса модуля взаимодейтсвия с базой данных

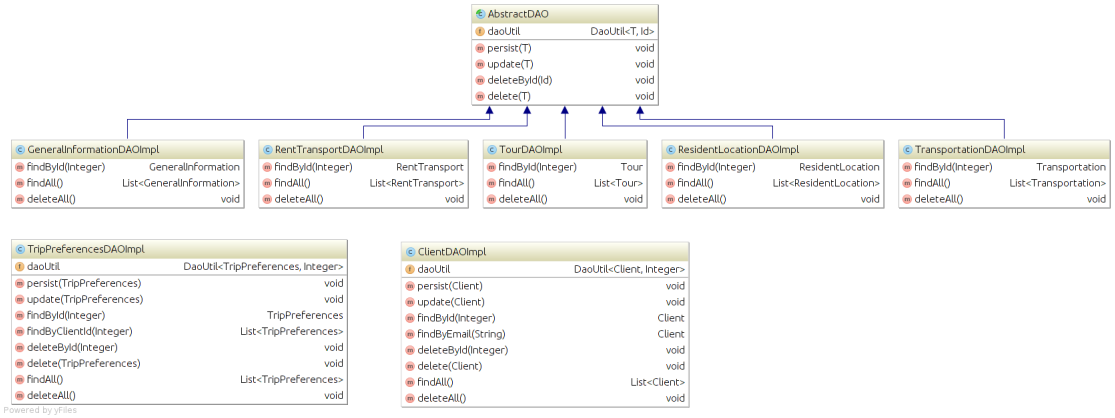

Рис. 11: Реализацаия внешнего интерфейса

4.3.7. Утилиты системы

В системе также реализованны 2 вспомогательных класса для упрощения взаимодейтсвия модулей.

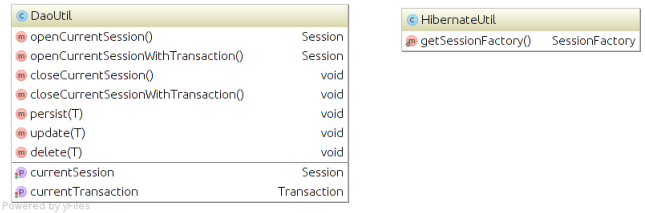

Рис. 12: Утилиты системы

DateTimeHelper - класс, выполняющий конвертирвоание между строковыми значниями даты и программными.

TourContentData - класс, осущесвтляющий загрузку и последующую подргузку (при необходимости) информации в систему из файла свойств приложения.

4.3.8. Исключения приложения

Для оповещения программных модулей и информировании пользователей о некорректной работе системы был разработан набор специализированных java-классов - исключений. В данной диаграмме приведены классы, позволяющие отлавливать такие исключительные ситуации как некорректный ввод пользовательских данных, ошибка при работе с GDS сервисами, отсутствие данного пользователя в системе.

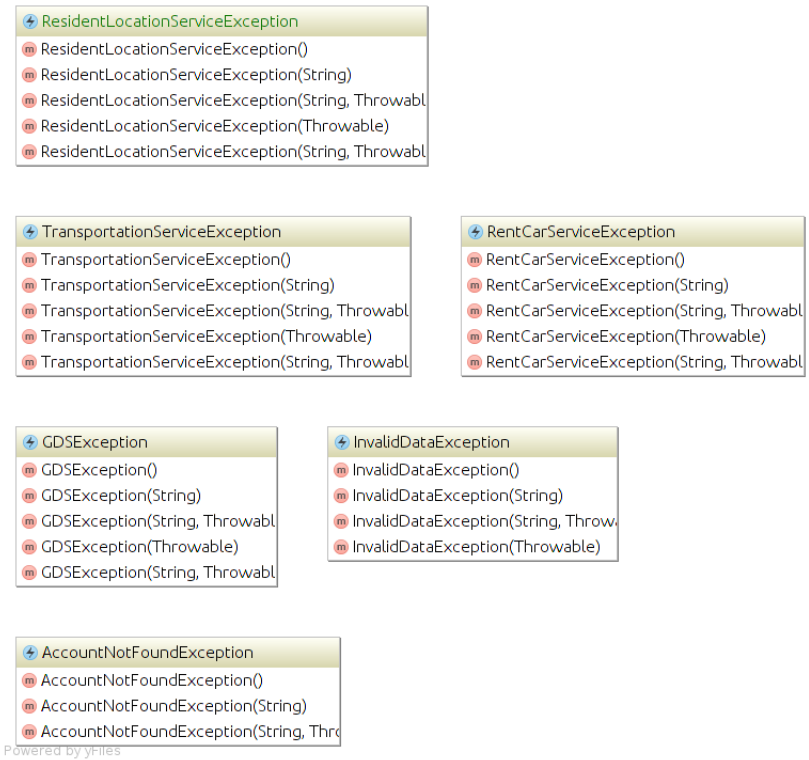

Рис. 13: Исключения

Разработаенное приложение работает лишь с данным набором исключений. Все ошибки, возникающие в стандартных или сторонних библиотеках заварачиваются в один из приведенных выше классов, что обеспечивает независимость системы от сторонних компонент и хорошую маштабируемость.

Примером может стать следующий фрагмент кода, в котором при несовпадении пароля, введенного пользователем, создается исключение InvalidDataException и выполняется пересылка запроса на страницу ошибки.

```
//password verification

if ( !password.equals(client.getPassword()) ){

InvalidDataException ex = new InvalidDataException(INCORRECT_PASS_OR_EMAIL_ERROR_MESSAGE);
request.setAttribute(”exception”, ex);
getServletContext().getRequestDispatcher(”/error.jspx”).forward(request, response);
return;
}
```

4.3.9. Взаимодействие системы с пользователем

Для получения и проверки данных пользователя в приложении реализован набор специализированных java-классов - сервлетов. В разработанной системе сервлеты взаимодействуют с пользователем посредством HTTP.

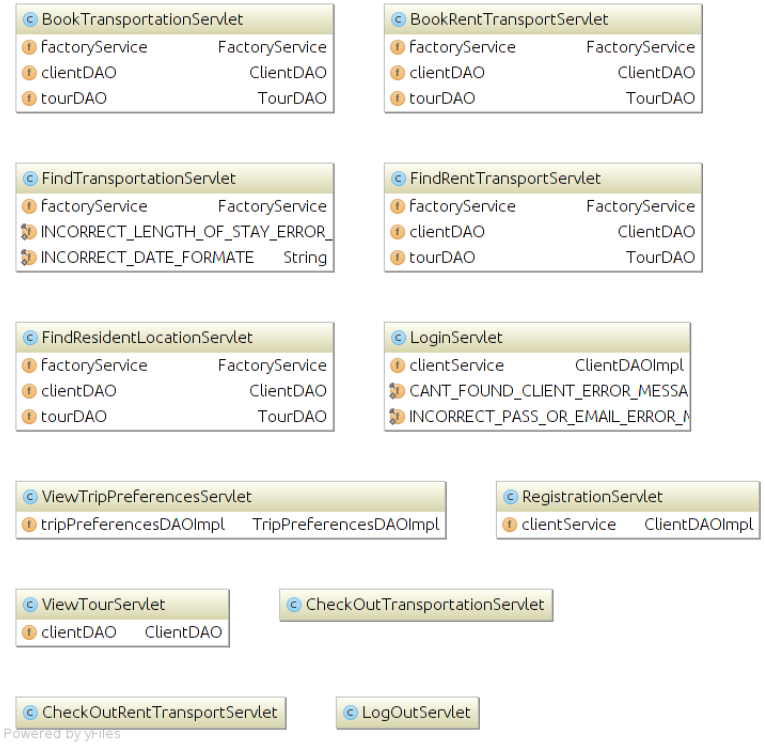

Рис. 14: Основные сервлеты

На данной диаграмме приведены основные классы, выполняющие регистрацию пользователей, вход в систему, заказ услуг и поиск туристического контента.

4.3.10. Модуль взаимодействия с GDS

Основной частью данного приложения является модуль для работы с глобальными распределительными системами. В данном модуле определено перечисление controller.gds.NameGDS имеющие 2 значения:

- SABRE - соответствует GDS Sabre;

- MY_WEB_SERVICE - веб-сервис-заглушка, разработанный для моделирования заказа контента.

Данный класс предназначен для задания имен основных внешних сервисов, с которыми будет работать приложение.

Он используется классом controller.gds.FactoryService.

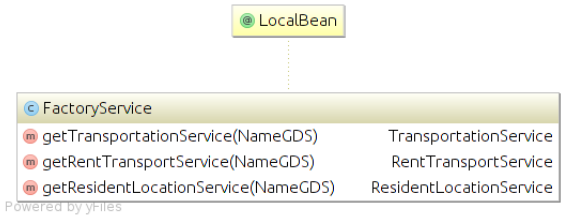

Рис. 15: Фабрика сервисов

В этом классе, реализованы методы, возвращающие экземпляры классов для работы с внешними сервисами соответствующие переданному перечислению controller.gds.NameGDS. Данное решение позволяет расширять приложение, добавляя новые внешние сервисы и соответстующие им значения перечисления controller.gds.NameGDS и даписывая логику фабрики сервисов.

Взаимодейсвие остальных модулей приложения с компонентом работы с GDS осуществляется через внешний интерфейс этого компонента, предствленного ниже.

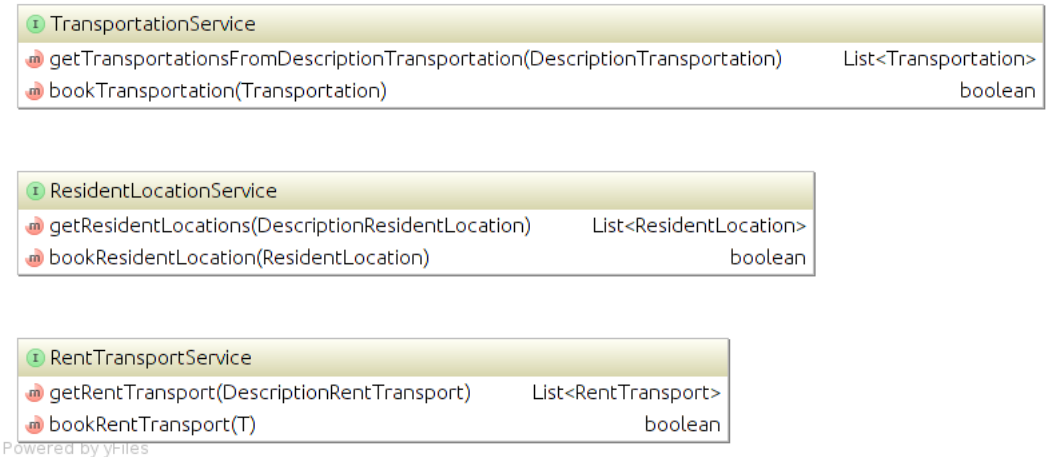

Рис. 16: Внешний интерфейс модуля для рыботы с GDS

Интерфейс позволяет получать списки классов, соответствующих различному контенту, также осуществлять заказ услуг.

В разработанном приложении в отдельный пакет constroller.gds.sabre была вынесена реализация данного интерфейса для глобальной распределительной системы Sabre.

В данном модуле кроме реализации внешнего интерфейса добавлен класс constroller.gds.sabre.SabreProperties, который позволяет получать необходимые служебные данные для взаимодействия с внешним сервисом. В текущей реализации информация поступает из обычного файла, но выбранное решение позволяет указывать любой источник данных и гарантирует корректную работу приложения.

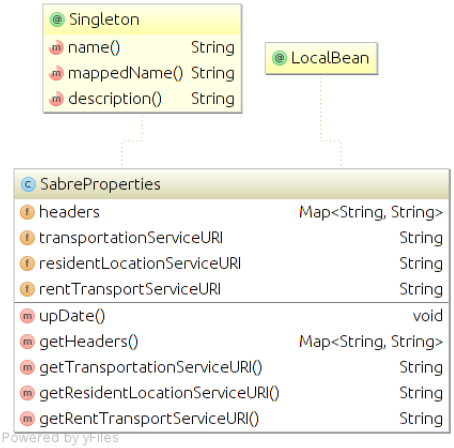

Рис. 17: Служебный класс SabreProperties

4.4. Функционал приложения

В результате разработанное приложение позволяет пользователю

- выполнять регистрацию и авторизацию;
- выполнять поиск отелей, перелетов и транспорта для аренды;
- выводить текущую информацию о контенте тура;
- выполнять заказ данного контента;

Скриншоты всех вышеперечисленных действий находятся в приложении к данной бакалаврской работе.

5. Тестирование сервиса

При разработке приложения было написано 26 модульных тестов, покрывающих полностью функционал приложения. В качестве фреймворка для тестирования использовался Junit 4 [5]. Ниже приведен фрагмент кода одного из модульных тестов, проверяющих корректность работы с БД.

```
/**
     * Данный тест выполняет проверку корректности сохранения и удаления Transportation
     * А также валидность сохраненных полей экземпляра Transportation
     * @throws ParseException
     */
    @Test
    public void CRUDTest() throws ParseException {

        //Удаляем все существующие записи в таблице
        transportationDAO.deleteAll();
        //создаем эезкмпляр для тестирования
        Transportation transportation = createTransportation();
        //сохраняем в БД
        transportationDAO.persist(transportation);

        //Достаем из БД все записи
        List<Transportation> transportationList = transportationDAO.findAll();

        // проверка, что лист содержит ровно одну запись
        assertEquals(1, transportationList.size());

        Transportation transportationRet = transportationList.get(0);
        //удаляем все записи и делаем проверку, что лист записей будет пустой
        transportationDAO.deleteAll();
        transportationList = transportationDAO.findAll();
        assertEquals(0, transportationList.size());

        //Проверяем обязательные поля вернувшегося экземпляра с начальным
        assertEquals(transportation.getCurrencyCode(), transportationRet.getCurrencyCode());
        assertEquals(transportation.getDepartureDateFrom(), transportationRet.getDepartureDateFrom());
        assertEquals(transportation.getDepartureDateTo(), transportationRet.getDepartureDateTo());
        assertEquals(transportation.getTransportKind(), transportationRet.getTransportKind());
        assertEquals(transportation.isChosen(), transportationRet.isChosen());
        assertEquals(transportation.getOriginLocation(), transportationRet.getOriginLocation());
        assertEquals(transportation.getDestinationLocation(), transportationRet.getDestinationLocation());
        assertEquals(true,  transportationRet.getPrice().compareTo(transportation.getPrice()) == 0);
    }
```

6. Заключение

В ходе работы были проанализированы средства для планирования и организации путешествий. Было разработано клиент-серверное приложение, позволяющее настраивать основные компоненты путешествия самостоятельно и находить по ним актуальные туристические предложения.

Ниже перечислены результаты данной работы:

- Проанализированы существующие решения;
- выбран оптимальный сервис для сбора информации;
- выбраны соответствующие инструменты для реализации;
- спроектирована структура базы данных;
- разработана общая архитектура приложения;
- реализована серверная и клиентская часть;
- протестирована система.

Приложение

Пользовательский интерфейс

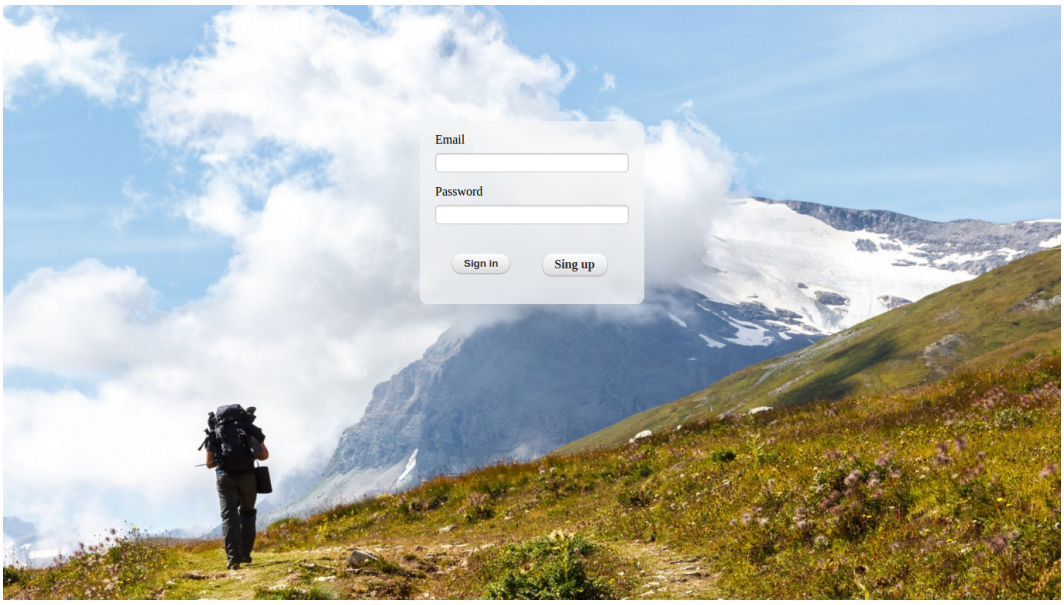

Рис. 18: Начальная страница

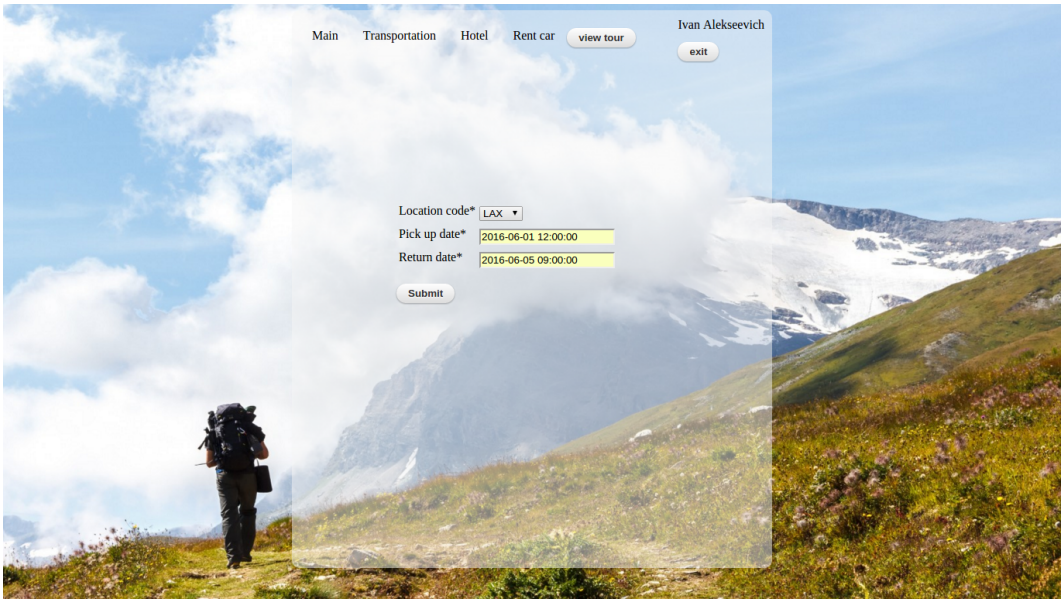

Рис. 19: Поиск транспорта для аренды

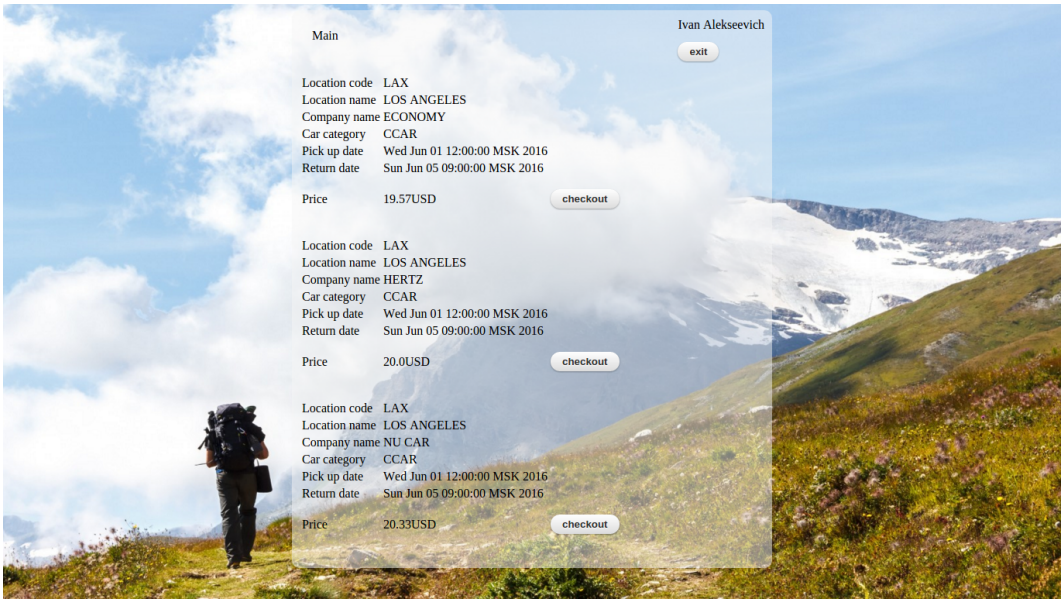

Рис. 20: Выбор услуги

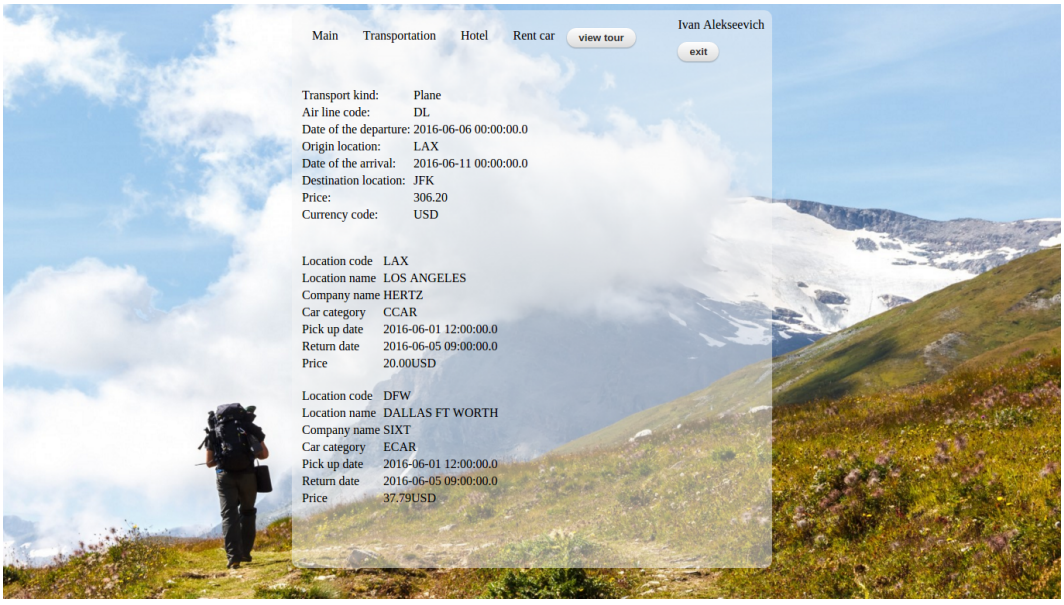

Рис. 21: Информация о туре

Фрагмент кода модуля взаимодействия с GDS

```
/*
    Метод получает на вход описание транспортного контента в виде экзмемпляра класса DescriptionTransportation
    Подключается к сервису, запрашивает информацию по данному набору данных и возвращает список даных,
    описывающих параметры поездки.
    Если ничего не найдено или возникла ошибка при взаимодействии с сревсисом выкидывает
    TransportationServiceException  с описанием проблемы
     */
    public List<Transportation> getTransportationsFromDescriptionTransportation(DescriptionTransportation dt)
    throws TransportationServiceException {

        // TEST BEGIN
//        SabreProperties sabreProperties = null;
//        try {
//            sabreProperties = new SabreProperties();
//        } catch (IOException e) {
//            e.printStackTrace();
//        }
        // TEST END

        /*
        Создаем экземпляр, соответствующий URI для сервиса
        Получаем карту заголовков из бина sabreProperties
         */
        String webTargetURI = sabreProperties.getTransportationServiceURI();
        WebTarget flightResource = client.target(webTargetURI);
        MultivaluedMap<String, Object> headers = new MultivaluedHashMap<String,
        Object>(sabreProperties.getHeaders());
        System.out.println();

        /*
        Устанавливем параметры запроса. Тип запроса и заголовки
         */
        Invocation.Builder builder = flightResource.
                queryParam(''origin'', dt.getOriginCode()).
                queryParam(''destination'', dt.getDestinationCode()).
                queryParam(''lengthofstay'', dt.getLengthOfStay()).
                queryParam(''departuredate'', sdf.format(dt.getDepartDate())).
                queryParam(''pointofsalecountry'', dt.getPointOfSaleCode()).
                request(MediaType.TEXT_PLAIN).
                headers(headers);

        // Выполняем запрос и получаем статус запроса и ответ в виде строки
        Response response = builder.get();
        int status = response.getStatus();
        String responseStr = response.readEntity(String.class);

        // Если статус не OK (200) бросаем исключение с описанием ошибки
        if (status != STATUS_OK){
            throw new TransportationServiceException(''Ошибка при выполнении запроса \n'' +
                                                     ''status: '' + status +
                                                     ''\nresponse: '' + responseStr);
        }
```

```
    /*
    Парсим ответ в список Transportation.
     При возникновении ошибок парсеров генерируем исключение с описанием ошибки
    */
    List<Transportation> resList = null;
    try {
        resList = getTransportationFromString(responseStr);
    } catch (ParseException e) {
        throw new TransportationServiceException(SABRE_ERROR);
    } catch (org.json.simple.parser.ParseException e) {
        throw new TransportationServiceException(SABRE_ERROR);
    }

    return resList;
}

/*
Метод парсит полученную строку от сервиса SABRE к списку Transportation
зная структуру получеенного json
 */
private static List<Transportation> getTransportationFromString(String inStr)
throws ParseException, org.json.simple.parser.ParseException {

    //Парсим входную строку в JSON
    JSONObject transpJson = (JSONObject) JSONValue.parseWithException(inStr);

    /*
    Зная структуру возвращаемого JSON от сервиса SABRE
    парсим его и достаем необходимую информацию для создания списка Transportation
     */
    String originCode = (String) transpJson.get(''OriginLocation'');
    String destinationCode = (String) transpJson.get(''DestinationLocation'');

    JSONObject fareInfo = (JSONObject)  ((JSONArray) transpJson.get(''FareInfo'')).get(0);

    String currencyCode = (String) fareInfo.get(''CurrencyCode'');
    String departureDateStr = (String) fareInfo.get(''DepartureDateTime'');
    String returnDateStr = (String) fareInfo.get(''ReturnDateTime'');

    JSONObject lowestFare = (JSONObject) fareInfo.get(''LowestFare'');
    String airLineCode = (String) ( (JSONArray) lowestFare.get(''AirlineCodes'')).get(0);
    Double fare = (Double) lowestFare.get(''Fare'');

    // Создаем список для возврата результата и Transportation
    List<Transportation> resList = new LinkedList<Transportation>();
    Transportation newTransportation = new Transportation();

    // Заполняем поля возвращаемого объекта
    newTransportation.setId(0);
    newTransportation.setTransportKind(''Plane'');
    newTransportation.setChosen(false);
    newTransportation.setCurrencyCode(currencyCode);
    newTransportation.setDestinationLocation(destinationCode);
    newTransportation.setPrice(BigDecimal.valueOf(fare));
    newTransportation.setOriginLocation(originCode);
```

```
        newTransportation.setAirlineCode(airLineCode);

        // Парсим дату
        newTransportation.setDepartureDateTo(sdf.parse(departureDateStr));
        newTransportation.setDepartureDateFrom(sdf.parse(returnDateStr));

        resList.add(newTransportation);

        return resList;
    }
```

Список литературы

[1] Elliott James O'Brien Timothy M. Fowler Ryan. Harnessing Hibernate. — 2008.

[2] Gupta Arun. Java EE 6 Pocket Guide. — 2012.

[3] Metsker Steven John. The Design Patterns Java Workbook. — 2002.

[4] Tim O'Brien Manfred Moser et al. Maven: The Complete Reference. — 2011.

[5] Vincent Massol Ted Husted. JUnit in Action. — 2003.

[6] В.С. Новиков. Инновации в туризме. — 2007.

[7] Лёшкин А.В. Кормалев Д.А. Анализ программных продуктов и решений для туристического бизнеса. — 2010.

[8] Эрих Гамма Ричард Хелм и др. Приемы объектно-ориентированного проектирования. Паттерны проектирования. — 2015.

[9] Ю. Примак Т. Исследование потенциала современных каналов дистрибуции сегментов туристических услуг. — Вісник ДІТБ, 2014.